Network Working Group
Internet Draft
Expiration Date: December 1999

Lou Berger
LabN Consulting

Der-Hwa Gan
Juniper Networks, Inc.

George Swallow
Cisco Systems, Inc.

Ping Pan
Bell Labs, Lucent

June 1999

# RSVP Refresh Reduction Extensions

draft-berger-rsvp-refresh-reduct-03.txt

## Status of this Memo



## Abstract

This document describes a number of mechanisms that reduce the refresh overhead of RSVP. The extensions can be used to reduce processing requirements of refresh messages, eliminate the state synchronization latency incurred when an RSVP message is lost and, when desired, suppress the generation of refresh messages. An extension to support detection of when an RSVP neighbor resets its state is also defined. These extension present no backwards

compatibility issues.

Contents


1 Introduction and Background ..... 4
1.1 Trigger and Refresh Messages ..... 5
2 RSVP Bundle Message ..... 5
2.1 Bundle Header ..... 6
2.2 Message Formats ..... 7
2.3 Sending RSVP Bundle Messages ..... 7
2.4 Receiving RSVP Bundle Messages ..... 8
2.5 Forwarding RSVP Bundle Messages ..... 9
2.6 Bundle-Capable Bit ..... 9
3 MESSAGE_ID Extension ..... 10
3.1 MESSAGE_ID Object ..... 11
3.2 Ack Message Format ..... 13
3.3 MESSAGE_ID Object Usage ..... 13
3.4 MESSAGE_ID ACK Object Usage ..... 16
3.5 Multicast Considerations ..... 16
3.5.1 Reference RSVP/Routing Interface ..... 18
3.6 Compatibility ..... 18
4 Summary Refresh Extension ..... 19
4.1 Srefresh Message Format ..... 20
4.2 Srefresh Message Usage ..... 21
4.3 Srefresh NACK ..... 22
4.4 Compatibility ..... 23
5 Hello Extension ..... 23
5.1 Hello Message Format ..... 24
5.2 HELLO Object ..... 25

5.3 Hello Message Usage ..... 26
5.4 Multi-Link Considerations ..... 27
5.5 Compatibility ..... 27
6 Reference Exponential Back-Off Procedures ..... 28
6.1 Outline of Operation ..... 28
6.2 Time Parameters ..... 28
6.3 Example Retransmission Algorithm ..... 29
7 Acknowledgments ..... 30
8 Security Considerations ..... 30
9 References ..... 31
10 Authors' Addresses ..... 32

## 1. Introduction and Background

The resource requirements (in terms of CPU processing and memory) for running RSVP on a router increases proportionally with the number of sessions. Supporting a large number of sessions can present scaling problems.

This document describes mechanisms to help alleviate one set of scaling issues. RSVP Path and Resv messages must be periodically refreshed to maintain state. The approach described effectively reduces the volume of messages which must be periodically sent and received, as well as the resources required to process refresh messages.

The described mechanisms also address issues of latency and reliability of RSVP Signaling. The latency and reliability problem occurs when a non-refresh RSVP message is lost in transmission. Standard RSVP [RFC2205] maintains state via the generation of RSVP refresh messages. In the face of transmission loss of RSVP messages, the end-to-end latency of RSVP signaling is tied to the refresh interval of the node(s) experiencing the loss. When end-to-end signaling is limited by the refresh interval, the establishment or change of a reservation may be beyond the range of what is acceptable for some applications.

One way to address the refresh volume problem is to increase the refresh period, "R" as defined in section 3.7 of [RFC2205]. Increasing the value of R provides linear improvement on transmission overhead, but at the cost of increasing the time it takes to synchronize state.

One way to address the latency and reliability of RSVP Signaling is to decrease the refresh period R. Decreasing the value of R provides increased probability that state will be installed in the face of message loss, but at the cost of increasing refresh message rate and associated processing requirements.

An additional issue is the time to deallocate resources after a tear message is lost. RSVP does not retransmit ResvTear or PathTear messages. If the sole tear message transmitted is lost, then resources will only be deallocated once the "cleanup timer" interval has passed. This may result in resources being allocated for an unnecessary period of time. Note that adjusting the refresh period has no impact on this issues since tear messages are not retransmitted.

The extensions defined in this document address both the refresh volume and the reliability issues with mechanisms other than adjusting refresh rate. A Bundle message is defined to reduce overall message

handling load. A Message_ID object is defined to reduce refresh message processing by allowing the receiver to more readily identify an unchanged message. A Message_ACK object is defined which can be used to detect message loss and, when used in combination with the Message_ID object, can be used to suppress refresh messages altogether. A summary refresh message is defined to enable refreshing state without the transmission of whole refresh messages, while maintaining RSVP's ability to indicate when state is lost or when next hops change. Finally, a hello protocol is defined to allow detection of the loss of a neighbor node or a reset of it's RSVP state information.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 1.1. Trigger and Refresh Messages

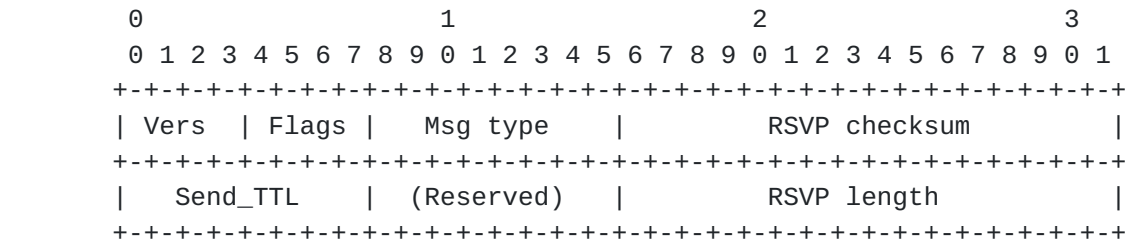
This document categorizes RSVP messages into two types: trigger and refresh messages. Trigger messages are those RSVP messages that advertise state or any other information not previously transmitted. Trigger messages include messages advertising new state, a route change that altered the reservation paths, or a reservation modification by a downstream router. Trigger messages also include those messages that include changes in non-RSVP processed objects, such as changes in the Policy or ADSPEC objects.

Refresh messages represent previously advertised state and contain exactly the same objects and same information as a previously transmitted message. Only Path and Resv messages can be refresh messages. Refresh messages are typically bit for bit identical to the corresponding previously transmitted message, with the exception of the flags in the MESSAGE_ID object. These flags allowed to differ in refresh messages.

## 2. RSVP Bundle Message

An RSVP Bundle message consists of a bundle header followed by a body consisting of a variable number of standard RSVP messages. A Bundle message is used to aggregated multiple RSVP messages within a single PDU. The term "bundling" is used to avoid confusion with RSVP reservation aggregation. The following subsections define the formats of the bundle header and the rules for including standard RSVP messages as part of the message.

### 2.1. Bundle Header




The format of the bundle header is identical to the format of the RSVP common header [RFC2205]. The fields in the header are as follows:

Vers: 4 bits

Protocol version number. This is version 1.

Flags: 4 bits

0x01: Bundle capable

If set, indicates to RSVP neighbors that this node is willing and capable of receiving bundle messages. This bit is meaningful only between adjacent RSVP neighbors.

0x02-0x08: Reserved

Msg type: 8 bits

12 = Bundle

RSVP checksum: 16 bits

The one's complement of the one's complement sum of the entire message, with the checksum field replaced by zero for the purpose of computing the checksum. An all-zero value means that no checksum was transmitted. Because individual sub-messages carry their own checksum as well as the INTEGRITY object for authentication, this field MAY be set to zero.

Send_TTL: 8 bits

The IP TTL value with which the message was sent. This is used by RSVP to detect a non-RSVP hop by comparing the IP TTL that a Bundle message sent to the TTL in the received message.

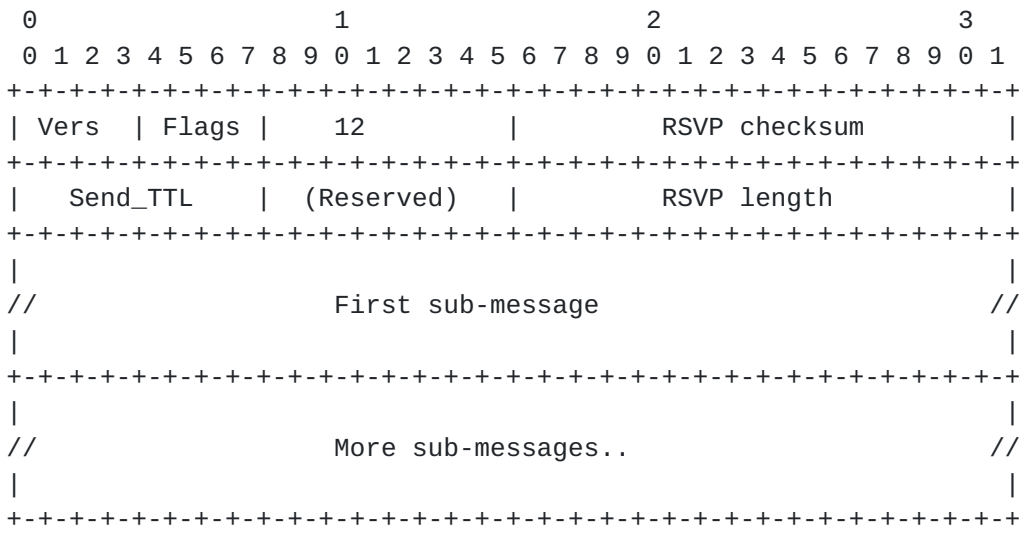
RSVP length: 16 bits

The total length of this RSVP Bundle message in bytes, including the bundle header and the sub-messages that follow.

## 2.2. Message Formats

An RSVP Bundle message must contain at least one sub-message. A sub-message MAY be any message type except for another Bundle message. Current valid sub-messages are RSVP Path, PathTear, PathErr, Resv, ResvTear, ResvErr, ResvConf, Ack or Hello messages.

Empty RSVP Bundle messages SHOULD NOT be sent. A Bundle message MUST NOT include another RSVP Bundle message as a sub-message.




## 2.3. Sending RSVP Bundle Messages

RSVP Bundle messages are sent hop by hop between RSVP-capable neighbors as "raw" IP datagrams with protocol number 46. Raw IP datagrams are also intended to be used between an end system and the first/last hop router, although it is also possible to encapsulate RSVP messages as UDP datagrams for end-system communication that cannot perform raw network I/O.

RSVP Bundle messages MUST not be used if the next hop RSVP neighbor does not support RSVP Bundle messages. Methods for discovering such information include: (1) manual configuration and (2) observing the Bundle-capable bit (see the description that follows) in the received RSVP messages. If the next hop RSVP neighbor is not known or changes in next hops cannot be identified via routing, Bundle messages MUST

NOT be sent. Note that when the routing next hop is not RSVP capable it will typically not be possible to identify changes in next hop.

Support for RSVP Bundle messages is optional. While message bundling helps in scaling RSVP, and in reducing processing overhead and bandwidth consumption, a node is not required to transmit every standard RSVP message in a Bundle message. A node MUST always be ready to receive standard RSVP messages.

The IP source address is local to the system that originated the Bundle message. The IP destination address is the next hop node for which the sub-messages are intended. These addresses need not be identical to those used if the sub-messages were sent as standard RSVP messages.

For example, the IP source address of Path and PathTear messages is the address of the sender it describes, while the IP destination address is the DestAddress for the session. These end-to-end addresses are overridden by hop-by-hop addresses while encapsulated in a Bundle message. These addresses can easily be restored from the SENDER_TEMPLATE and SESSION objects within Path and PathTear messages. For Path and PathTear messages, the next hop node can be identified either via a received ACK or from a received corresponding Resv message. Path and PathTear messages for multicast sessions MUST NOT be sent in Bundle messages except when the outgoing interface is a point-to-point interface and it is known that the next hop is RSVP capable.

RSVP Bundle messages SHOULD NOT be sent with the Router Alert IP option in their IP headers. This is because Bundle messages are addressed directly to RSVP neighbors.

Each RSVP Bundle message MUST occupy exactly one IP datagram. If it exceeds the MTU, the datagram is fragmented by IP and reassembled at the recipient node. A single RSVP Bundle message MUST NOT exceed the maximum IP datagram size, which is approximately 64K bytes.

## 2.4. Receiving RSVP Bundle Messages

If the local system does not recognize or does not wish to accept an Bundle message, the received messages shall be discarded without further analysis.

The receiver next compares the IP TTL with which a Bundle message is sent to the TTL with which it is received. If a non-RSVP hop is detected, the number of non-RSVP hops is recorded. It is used later in processing of sub-messages.

Next, the receiver verifies the version number and checksum of the RSVP Bundle message and discards the message if any mismatch is found.

The receiver then starts decapsulating individual sub-messages. Each sub-message has its own complete message length and authentication information. Each sub-message is processed per standard RSVP.

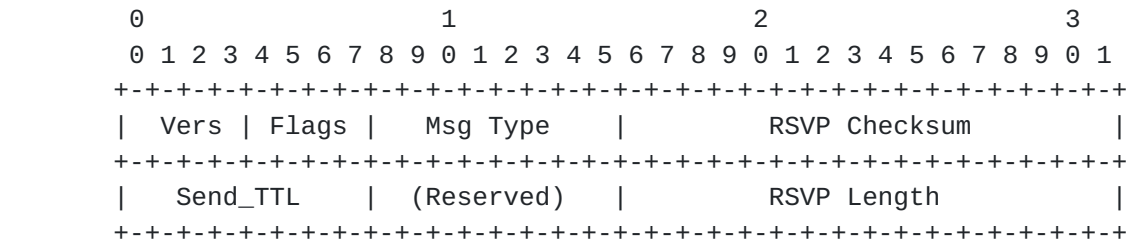
## 2.5. Forwarding RSVP Bundle Messages

When an RSVP router receives a Bundle messages which is not addressed to one of it's IP addresses, it SHALL forward the message. Non-RSVP routers will treat RSVP Bundle messages as any other IP datagram.

When individual sub-messages are being forwarded, they MAY be encapsulated in another Bundle message before sending to the next hop neighbor. The Send_TTL field in the sub-messages should be decremented properly before transmission.

## 2.6. Bundle-Capable Bit

To support message bundling, an additional capability bit is added to the common RSVP header, which is defined in [RFC2205].




Flags: 4 bits

0x01: Bundle capable

If set, indicates to RSVP neighbors that this node is willing and capable of receiving Bundle messages. This bit is meaningful only between adjacent RSVP neighbors.

## 3. MESSAGE_ID Extension

Two new objects are defined as part of the MESSAGE_ID extension. The two object types are the MESSAGE_ID object and the MESSAGE_ID ACK object. The objects are used to support acknowledgments and, when used in conjunction with the Hello Extension described in Section 5, to indicate when refresh messages are not needed after an acknowledgment. When refreshes are normally generated, the MESSAGE_ID object can also be used to simply provide a shorthand indication of when a message represents new state. Such information can be used on the receiving node to reduce refresh processing requirements.

Message identification and acknowledgment is done on a hop-by-hop basis. Acknowledgment is handled independent of SESSION or message type. Both types of MESSAGE_ID objects contain a message identifier. The identifier MUST be unique on a per source IP address basis across messages sent by an RSVP node and received by a particular node. No more than one MESSAGE_ID object may be included in an RSVP message. Each message containing an MESSAGE_ID object may be acknowledged via a MESSAGE_ID ACK object. MESSAGE_ID ACK objects may be sent piggy-backed in unrelated RSVP messages or in RSVP Ack messages.

Either type of MESSAGE_ID object may be included in a bundle sub-message. When included, the object is treated as if it were contained in a standard, unbundled, RSVP message. Only one MESSAGE_ID object MAY be included in a (sub)message and it MUST follow any present MESSAGE_ID ACK objects. When no MESSAGE_ID ACK objects are present, the MESSAGE_ID object MUST immediately follow the INTEGRITY object. When no INTEGRITY object is present, the MESSAGE_ID object MUST immediately follow the (sub)message header.

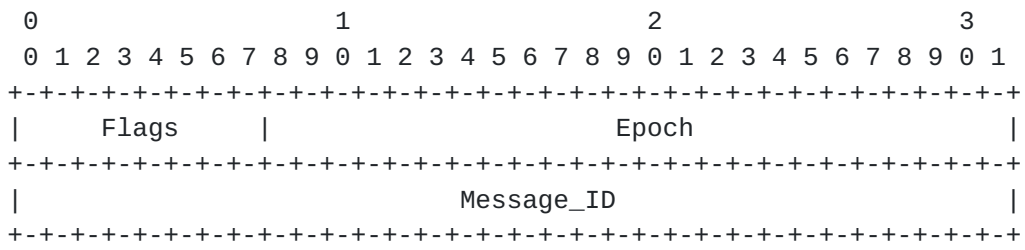
When present, one or more MESSAGE_ID ACK objects MUST immediately follow the INTEGRITY object. When no INTEGRITY object is present, the MESSAGE_ID ACK objects MUST immediately follow the the (sub)message header. An MESSAGE_ID ACK object may only be included in a message when the message's IP destination address matches the unicast address of the node that generated the message(s) being acknowledged.

### 3.1. MESSAGE_ID Object

MESSAGE_ID Class = 166 (Value to be assigned by IANA of form 10bbbbbb)

MESSAGE_ID object

Class = MESSAGE_ID Class, C_Type = 1




Flags: 8 bits

0x80 = Summary_Capable flag

Indicates that the sender supports the summary refresh extension. This flag MUST be set if the node supports the summary refresh extension. See Section 4.4 for description of handling by receiver.

0x40 = ACK_Desired flag

Indicates that the sender is willing to accept a message acknowledgment. Acknowledgments MUST be silently ignored when they are sent in response to messages whose ACK_Desired flag is not set. This flag MUST be set when the Last_Refresh flag is set.

0x20 = Last_Refresh flag

Used in Resv and Path refresh messages to indicate that the sender will not be sending further refreshes. When set, the ACK_Desired flag MUST also be set. This flag MUST NOT be set when the HELLO messages are not being exchanged with the neighboring RSVP node.

Epoch: 24 bits

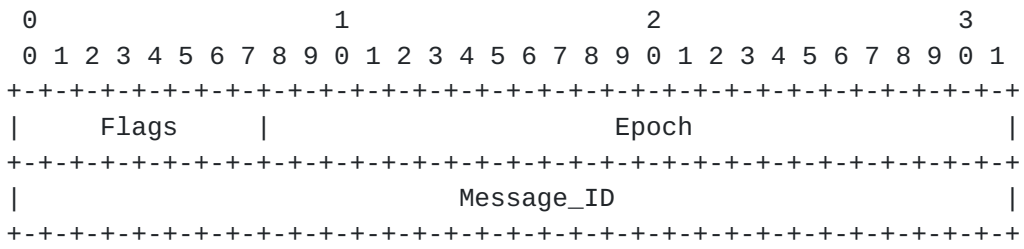
A value that indicates when the Message_ID sequence has reset. SHOULD be randomly generated each time a node reboots. This value MUST NOT be changed during normal operation.

Message_ID: 32 bits

When combined with the message generator's IP address, the Message_ID field uniquely identifies a message. This field is ordered and only decreases in value when the Epoch changes or the value wraps.

MESSAGE_ID ACK object

Class = MESSAGE_ID Class, C_Type = 2




Flags: 8 bits

0x80 = Summary_Capable flag

Indicates that the sender supports the summary refresh extension. This flag MUST be set if the node supports the summary refresh extension. See Section 4.4 for description of handling by receiver.

0x40 = Refresh_NACK flag

Indicates that no state was found corresponding to the indicated message identifier. This flag SHALL ONLY be set when the matching Epoch and Message_ID field values were received in a Summary Refresh message, and MUST NOT be set in response to a MESSAGE_ID object received in any other message. See Section 4 for details.

Epoch: 24 bits

The Epoch field copied from the message being acknowledged.

Message_ID: 32 bits

The Message_ID field copied from the message being acknowledged.

### 3.2. Ack Message Format

Ack messages carry one or more MESSAGE_ID ACK objects. They MUST NOT contain any MESSAGE_ID objects. Ack messages are sent hop-by-hop between RSVP nodes. The IP destination address of an Ack message is the unicast address of the node that generated the message(s) being acknowledged. For Path, PathTear, Resv, and RervErr messages this is taken from the RSVP_HOP Object. For PathErr and ResvErr messages this is taken from the message's source address. The IP source address is an address of the node that sends the Ack message.

The Ack message format is as follows:

```
<ACK Message> ::= <Common Header> [ <INTEGRITY> ]
                      <MESSAGE_ID ACK>
                      [ <MESSAGE_ID ACK> ... ]
```

For Ack messages, the Msg Type field of the Common Header MUST be set to 13 (Value to be assigned by IANA).

### 3.3. MESSAGE_ID Object Usage

The MESSAGE_ID object may be included in any RSVP message other than the Ack message. The MESSAGE_ID object is always generated and processed hop-by-hop. The IP address of the object generator is represented in a per RSVP message type specific fashion. For Path and PathTear messages the generator's IP address is contained in the RSVP_HOP. For Resv, ResvTear, PathErr, ResvErr, ResvConf and Bundle messages the generator's IP address is the source address in the IP header.

The Epoch field contains a generator selected value. The value is used to indicate when the sender resets the values used in the Message_ID field. This information is used by the receiver to detect out of order messages. On startup, a node SHOULD randomly select a value to be used in the Epoch field. The node SHOULD ensure that the selected value is not the same as was used when the node was last operational. The value MUST NOT be changed unless the node or the RSVP agent is restarted.

The Message_ID field contains a generator selected value. This value, when combined with the generator's IP address, identifies a particular RSVP message and the specific state information it represents. When a node is sending a refresh message with a MESSAGE_ID object, it SHOULD use the same Message_ID value that was used in the RSVP message that first advertised the state being refreshed. When a node is sending a trigger message, the Message_ID value MUST have a

value that is greater than any other previously used value. A value is considered to have been used when it has been sent in any message using the associated IP address. Note that this 32-bit value MAY wrap.

The ACK_Desired flag is set when the MESSAGE_ID object generator is capable of accepting MESSAGE_ID ACK objects. Such information can be used to ensure reliable delivery of error and confirm messages and to support fast refreshes in the face of network loss. Nodes setting the ACK_Desired flag SHOULD retransmit unacknowledged messages at a more rapid interval than the standard refresh period until the message is acknowledged or until a "rapid" retry limit is reached. Rapid retransmission rate SHOULD be based on well known exponential back-off procedures. See Section 6 for details on one exponential back-off retransmission approach. Note that nodes setting the ACK_Desired flag for unicast sessions, do not need to track the identify of the next hop since all that is expected is an ACK, not an ACK from a specific next hop. Issues relate to multicast sessions are covered in a later section. The ACK_Desired flag will typically be set only in trigger messages. The ACK_Desired flag MAY be set in refresh messages.

The Last_Refresh flag may be set in Path and Resv messages when the MESSAGE_ID object generator is exchanging Hello messages, described in Section 5, with the next hop RSVP node. When a refresh message with the Last_Refresh flag set is generated, normal refresh generation MUST continue until the message containing the Last_Refresh flag is acknowledged. Although, messages removing state advertised in such messages MUST be retransmit until acknowledged or a maximum retry limit is reached in order to cover certain packet loss conditions. Messages removing state include PathTear and ResvTear.

When sending MESSAGE_ID objects with the Last_Refresh flag set, special care must be taken to properly advertise state. Specifically, refresh processing MUST continue per standard RSVP processing until after a acknowledgment is received. Suppression of refresh processing MAY ONLY occur after an acknowledgment is received for a MESSAGE_ID object with the Last_Refresh flag set. Note that the Last_Refresh flag MAY ONLY be set when the RSVP next hop is exchanging Hello messages with the message generator.

When a Path message for a new session arrives, the RSVP next hop may not always be known. When the RSVP next hop is not known, the Last_Refresh flag MUST NOT be set. Once the next hop of a unicast session is identified, only then may the Last_Refresh flag be set. (Issues relate to multicast sessions are covered in a later section.) There are several ways to identify the RSVP next hop of a new unicast session. Some are more conservative than other, e.g., waiting for a

Resv message versus checking if the other end of a PPP link supports Hello messages. Since there are no interoperability issues, the specific mechanism used to identify the RSVP next hop of a new session is a specific implementation choice. In most implementations, it is expected that an advertiser of Path state will do standard refresh processing until either an ACK is received for a Path message advertising a new session, or a corresponding Resv message is received.

Nodes processing incoming MESSAGE_ID objects SHOULD check to see if a newly received message is out of order and can be ignored. Out of order messages can be identified by examining the values in the Epoch and Message_ID fields. If the Epoch value differs from the value previously received from the message sender, the receiver MUST fully processes the message. If the Epoch values match and the Message_ID value is greater than the largest value previously received from the sender, the receiver MUST fully processes the message. If the value is less than the largest value previously received from the sender, then the receiver SHOULD check the value previously received for the state associated with the message. This check should be performed for the currently defined messages: Path, Resv, PathTear, ResvTear, PathErr and ResvErr. If no local state information can be associated with the message, the receiver MUST fully processes the message. If local state can be associated with the message and the received Message_ID value is less than the most recently received value associated with the state, the message SHOULD be ignored, i.e., silently dropped.

Nodes receiving messages containing MESSAGE_ID objects SHOULD use the information in the objects to aid in determining if a message represents new state or a state refresh. Note that state is only refreshed in Path and Resv messages. If the received Epoch values differs from the value previously received from the message sender, the message is a trigger message and the receiver MUST fully processes the message. If a Path or Resv message contains the same Message_ID value that was used in the most recently received message for the same session and, for Path messages, SENDER_TEMPLATE then the receiver SHOULD treat the message as a state refresh. If the Message_ID value is grater than the most recently received value, the receiver MUST fully processes the message. If the Message_ID value is less than the most recently received value, the receiver SHOULD ignore the message.

Nodes receiving a non-out of order message containing a MESSAGE_ID object with the ACK_Desired flag set, SHOULD respond with a MESSAGE_ID ACK object. If a node supports the Hello extension it MUST also check the Last_Refresh flag of received Resv and Path messages. If the flag is set, the receiver MUST NOT timeout state associated with associated message. The receiver MUST also be prepared to

properly process refresh messages. Messages containing a MESSAGE_ID ACK object with the Last_Refresh flag set MUST NOT be acknowledged when either the receiving node doesn't support the Hello extension or Hello messages aren't being exchanged with the message generator.

### 3.4. MESSAGE_ID ACK Object Usage

The MESSAGE_ID ACK object is used to acknowledge receipt of messages containing MESSAGE_ID objects that were sent with the ACK_Desired flag set. The Epoch and Message_ID fields of a MESSAGE_ID ACK object MUST have the same value as was received. A MESSAGE_ID ACK object MUST NOT be generated in response to a received MESSAGE_ID object when the ACK_Desired flag is not set, except as noted in Section 4.3.

A MESSAGE_ID ACK object MAY be sent in any RSVP message that has an IP destination address matching the generator of the associated MESSAGE_ID object. The MESSAGE_ID ACK object will not typically be included in the non hop-by-hop Path, PathTear and ResvConf messages. When no appropriate message is available, one or more MESSAGE_ID ACK objects SHOULD be sent in an Ack message. Implementations SHOULD include MESSAGE_ID ACK objects in standard RSVP messages when possible.

MESSAGE_ID ACK objects received with the Refresh_NACK flag set MUST process the object as described in Section 4.3. Upon receiving a MESSAGE_ID ACK object with the Refresh_NACK flag not set, a node SHOULD stop retransmitting the message at the "rapid" retry rate. If the received object also has the Last_Refresh flag set, normal refresh generation SHOULD be suppressed for the associated state. As previously mentioned, special care must be taken to properly advertise state when sending MESSAGE_ID objects with the Last_Refresh flag set, see section 3.3.

### 3.5. Multicast Considerations

Path and PathTear messages may be sent to IP multicast destination addresses. When the destination is a multicast address, it is possible that a single message containing a single MESSAGE_ID object will be received by multiple RSVP next hops. When the ACK_Desired flag is set in this case, acknowledgment processing is more complex. There are a number of issues to be addressed including ACK implosion, number acknowledgments to be expected and handling of new receivers.

ACK implosion occurs when each receiver responds to the MESSAGE_ID object at approximately the same time. This can lead to a potentially large number of MESSAGE_ID ACK objects being simultaneously

delivered to the message generator. To address this case, the receiver MUST wait a random interval prior to acknowledging a MESSAGE_ID object received in a message destined to a multicast address. The random interval SHOULD be between zero (0) and a configured maximum time. The configured maximum SHOULD be set in proportion to the refresh and "rapid" retransmission interval, i.e, such that the maximum back-off time does not result in retransmission.

A more fundamental issue is the number of acknowledgments that the upstream node, i.e., the message generator, should expect. The number of acknowledgments that should be expected is the same as the number of RSVP next hops. In the router-to-router case, the number of next hops can usually be obtained from routing. When hosts are either the upstream node or the next hops, the number of next hops will typically not be readily available. Another case where the number of RSVP next hops will typically not be known is when there are non-RSVP routers between the message generator and the RSVP next hops.

When the number of next hops is not known, the message generator SHOULD only expect a single response and MUST NOT set the Last_Refresh flag in MESSAGE_ID objects. The result of this behavior will be special retransmission handling until the message is delivered to at least one next hop, then followed by standard RSVP refreshes. Standard refresh messages will synchronize state with any next hops that don't receive the original message.

Another issue is handling new receivers. It is possible that after sending a Path message and handling of expected number of acknowledgments that a new receiver joins the group. In this case a new Path message must be sent to the new receiver. When normal refresh processing is occurring, there is no issue. When normal refresh processing is suppressed, a Path message must still be generated. In the router-to-router case, the identification of new next hops can usually be obtained from routing. When hosts are either the upstream node or the next hops, the identification of new next hops will typically not be possible. Another case where the identification of new RSVP next hops will typically not be possible is when there are non-RSVP routers between the message generator and the RSVP next hops.

When identification of new next hops is not possible, the message generator SHOULD only expect a single response and MUST NOT set the Last_Refresh flag in MESSAGE_ID objects. The result of this behavior will be special retransmission handling until the message is delivered to at least one next hop, then followed by standard RSVP refreshes. Standard refresh messages will synchronize state with any next hops that don't receive the original message either due to loss or not yet being a group member.

There is one additional minor issue with multiple next hops. The issue is handling a combination of standard-refresh and non-refresh next hops, i.e., Hello messages are being exchanged with some neighboring nodes but not with others. When this case occurs, refreshes MUST be generated per standard RSVP and the Last_Refresh flag MUST NOT be set.

### 3.5.1. Reference RSVP/Routing Interface

When using the MESSAGE_ID extension with multicast sessions it is preferable for RSVP to obtain the number of next hops from routing and to be notified when that number changes. The interface between routing and RSVP is purely an implementation issue. Since RSVP [RFC2205] describes a reference routing interface, we present a version of the RSVP/routing interface updated to provide number of next hop information. See [RFC2205] for previously defined parameters and function description.

- Route Query

```
Mcast_Route_Query( [ SrcAddress, ] DestAddress,
                    Notify_flag )
                    -> [ IncInterface, ] OutInterface_list,
                       NHops_list
```

- Route Change Notification

```
Mcast_Route_Change( ) -> [ SrcAddress, ] DestAddress,
                  [ IncInterface, ] OutInterface_list,
                  NHops_list
```

NHops_list provides the number of multicast group members reachable via each OutInterface_list entry.

## 3.6. Compatibility

There are no backward compatibility issues raised by the MESSAGE_ID Class. The MESSAGE_ID Class has an assigned value whose form is 10bbbbbb. Per RSVP [RFC2205], classes with values of this form must be ignored and not forwarded by nodes not supporting the class. When the receiver of a MESSAGE_ID object does not support the class, the object will be silently ignored. The generator of the MESSAGE_ID object will not see any acknowledgments and therefore refresh messages per standard RSVP. Lastly, since the MESSAGE_ID ACK object can only be issued in response to the MESSAGE_ID object, there are no possible issues with this object or Ack messages.

Implementations supporting Path and Resv state refresh suppression

via the MESSAGE_ID object's Last_Refresh flag MUST also support the Hello extension. Such implementations SHOULD initiate Hello processing and MUST be able to respond to Hello messages.

## 4. Summary Refresh Extension

The Summary Refresh extension enables the refreshing of RSVP state without the transmission of standard Path or Resv messages. The benefits of the described extension are that it reduces the amount of information that must be transmitted and processed in order to maintain RSVP state synchronization. Importantly, the described extension preserves RSVP's ability to handle non-RSVP next hops and to adjust to changes in routing. This extension cannot be used with Path or Resv messages that contain any change from previously transmitted messages, i.e, are not refresh messages.

The summary refresh extension uses the previously defined MESSAGE_ID object class, the ACK message, and a new Srefresh message. The new message carries a list of MESSAGE_ID objects corresponding to the Path and Resv states that are to be refreshed. An RSVP node receiving an Srefresh message, matches each received MESSAGE_ID object with installed Path or Resv state. All matching state is updated as if a normal RSVP refresh message is received. If matching state cannot be found, then the Srefresh message sender is notified via a refresh NACK.

Since Srefresh messages can carry multiple MESSAGE_ID objects, Srefresh messages are not expected to be sent in an RSVP aggregate messages. The flags field of MESSAGE_ID objects carried in Srefresh messages may be set.

A refresh NACK is indicated by setting the Refresh_NACK flag in the MESSAGE_ID ACK object. The rules for sending a MESSAGE_ID ACK object with the Refresh_NACK flag set are the same as was described in the previous section. This includes sending MESSAGE_ID ACK object both piggy-backed in unrelated RSVP messages or in RSVP ACK messages.

Nodes supporting the described extension can advertise their support and detect if an RSVP neighbor also supports the extension. This is accomplished via flag in the MESSAGE_ID class objects.

### 4.1. Srefresh Message Format

Srefresh messages carry one or more MESSAGE_ID objects. A single Srefresh message MAY refresh both Path or Resv state. Srefresh messages carrying MESSAGE_ID objects corresponding to Path state SHOULD be sent with a destination IP address equal to the address carried in the corresponding SESSION objects. The destination IP address MAY be set to the RSVP next hop when the next hop is known to be RSVP capable and the session is unicast or the outgoing interface is a point-to-point interface. Srefresh messages carrying MESSAGE_ID objects corresponding to Resv state MUST be sent with an destination IP address set to the Resv state's previous hop.

The source IP address of an Srefresh message is an address of the node that generates the message. The source IP address MUST match the addressed associate with the MESSAGE_ID objects when they were included in a standard RSVP message. As previously mentioned, the address associate with a MESSAGE_ID object is represented in a per RSVP message type specific fashion. For Path and PathTear messages the associated IP address is contained in the RSVP_HOP. For Resv, ResvTear, PathErr, ResvErr, ResvConf and Bundle messages the associated IP address is the source address in the IP header.

Srefresh messages that are sent destined to a session's destination IP address MUST be sent the Router Alert IP option in their IP headers. Srefresh messages addressed directly to RSVP neighbors SHOULD NOT be sent with the Router Alert IP option in their IP headers.

Each Srefresh message MUST occupy exactly one IP datagram. If it exceeds the MTU, the datagram is fragmented by IP and reassembled at the recipient node. A single RSVP Srefresh message MUST NOT exceed the maximum IP datagram size, which is approximately 64K bytes.

The Srefresh message format is as follows:

```
  <Srefresh Message> ::= <Common Header> [ <INTEGRITY> ]
                         <MESSAGE_ID>
                         [ <MESSAGE_ID> ... ]
```

For Srefresh messages, the Msg Type field of the Common Header MUST be set to 14 (Value to be assigned by IANA).

### 4.2. Srefresh Message Usage

An Srefresh message MAY be generated to refresh Resv or Path state. If an Srefresh message is used to refresh some particular state, then the generation of a standard refresh message SHOULD be suppressed. A state's refresh interval is not affected by the use of Srefresh message based refreshes. An Srefresh message MUST NOT be used in place of a trigger Path or Resv message, i.e., one that would advertise a state change.

When generating an Srefresh message, a node SHOULD refresh as much Path and Resv state as is possible by including as many MESSAGE_ID objects in the same Srefresh message. Only MESSAGE_ID objects that meet the previously described source and destination IP address restrictions may be included in the same Srefresh message. Identifying Resv state that can refreshed using the same Srefresh message is fairly straightforward. Identifying which Path state may be included is a little more complex.

Independent of the state being refreshed, only state that was previously advertised in Path and Resv messages containing MESSAGE_ID objects can be refreshed via an Srefresh message. Srefresh message based refreshes must also have the state synchronization properties of Path or Resv message based refreshes. Specifically, the use of Srefresh messages MUST NOT result in state being timed-out at the RSVP next hop. The period at which state is refreshed when using Srefresh messages MAY be shorter than the period that would be used when using Path or Resv message based refreshes, but it MUST NOT be longer. The particular approach used to trigger Srefresh message based refreshes is implementation specific. Some possibilities are triggering Srefresh message generation based on each state's refresh period or, on a per interface basis, periodically generating Srefresh messages to refresh all state that has not been refreshed within the state's refresh interval. Other approaches are also possible.

When generating an Srefresh message, there are two methods for identifying which Path state may be refreshed in a specific message. In both cases, the previously mentioned refresh interval and source IP address restrictions must be followed. The primary method is to include only those sessions that share the same destination IP address in the same Srefresh message. When using this method, the destination address of each session MUST be the same as the destination address in the IP header of the Srefresh message.

The secondary method for identifying which Path state may be refreshed within a single Srefresh message is an optimization. This method MAY be used when the next hop is known to support RSVP and either the session is unicast or the outgoing interface is a point-

to-point interface. This method MUST NOT be used when the next hop is not known to support RSVP or when the outgoing interface is to a multi-access network and the session is to a multicast address. When using this method, the destination address in the IP header of the Srefresh message is always the next hop's address. When the outgoing interface is a point-to-point interface, all Path state advertised out the interface SHOULD be included in the same Srefresh message. When the outgoing interface is not a point-to-point interface, all unicast session Path state SHOULD be included in the same Srefresh message.

Identifying which Resv state may be refreshed within a single Srefresh message is based simply on the source and destination IP addresses. Any state that was previously advertised in Resv messages with the same IP addresses as an Srefresh message MAY be included.

After identifying the Path and Resv state that can be included in a particular Srefresh message, the message generator adds to the message a MESSAGE_ID object matching each identified state's previously used object. Once the Srefresh message is composed, the message generator transmits the message out the proper interface.

Upon receiving an Srefresh message, a receiver MUST attempt to identify matching Path or Resv state. Matching is done based on the source address in the IP header of the Srefresh message and each MESSAGE_ID object contained in the message. For each received MESSAGE_ID object, the receiver performs an installed state lookup based on the values contained in the object. If matching state can be found, then the receiver MUST update the matching state information as if a standard refresh had been received. The receiver MUST also process the flags contained in the MESSAGE_ID object per Sections 3 and 4.4. If the receiver cannot identify any matching installed state, then a Srefresh NACK MUST be generated corresponding to the unmatched MESSAGE_ID object.

### 4.3. Srefresh NACK

Srefresh NACKs are used to indicated that a received MESSAGE_ID object does not match any installed state. An Srefresh NACK is encoded in a MESSAGE_ID ACK object with the Refresh_NACK flag set. When generating an Srefresh NACK, The epoch and Message_ID fields of a MESSAGE_ID ACK object MUST have the same value as was received. Objects with the Refresh_NACK flag set are transmitted as previously described, see Section 3.4.

MESSAGE_ID ACK objects received with the Refresh_NACK flag set indicate that the object generator does not have any installed state

matching the object. Upon receiving a MESSAGE_ID ACK object with the Refresh_NACK flag set, the receiver performs an installed Path or Resv state lookup based on the values contained in the object. If matching state is found, then the receiver MUST transmit the matching state via a standard Path or Resv message. If the receiver cannot identify any installed state, then no action is required.

### 4.4. Compatibility

Nodes supporting the Summary Refresh extension advertise their support via the Summary_Capable flag in all MESSAGE_ID call objects transmitted by the node. This enables supporting nodes to detect each other. When it is not known if a next hop supports the extension, standard Path and Resv message based refreshes MUST be used. Note that when the routing next hop does not support RSVP, it will not always be possible to detect if the RSVP next hop supports the Summary Refresh extension. Therefore, when the routing next hop is not RSVP capable the Srefresh message based refresh SHOULD NOT be used. A node MAY be administratively configured to use Srefresh messages in all cases when all RSVP nodes in a network are known to support the Summary Refresh extension.

Nodes supporting the Summary Refresh extension must also take care to recognize when a next hop stops sending MESSAGE_ID objects with the Summary_Capable flag set. To cover this case, nodes supporting the Summary Refresh extension MUST examine each received Summary_Capable flag. If the flag changes from indicating support to indicating non-support then Srefresh messages MUST NOT be used for subsequent state refreshes to that neighbor.

## 5. Hello Extension

The RSVP Hello extension enables RSVP nodes to detect a loss of a neighboring node's state information. In standard RSVP, such detection occurs as a consequence of RSVP's soft state model. When refresh message generation is suppressed via the previously discussed Last_Refresh flag processing, the Hello extension is needed to address this failure case. The Hello extensions is not intended to provide a link failure detection mechanism, particularly in the case of multiple parallel unnumbered links.

The Hello extension is specifically designed so that one side can use the mechanism while the other side does not. Neighbor RSVP state tracking may be initiated at any time. This includes when neighbors first learn about each other, or just when neighbors are sharing Resv or Path state. As previously stated, all implementations supporting

state refresh suppression are required to also support the Hello Extension.

The Hello extension is composed of a Hello message, a HELLO REQUEST object and a HELLO ACK object. Hello processing between two neighbors supports independent selection of, typically configured, failure detection intervals. Each neighbor can autonomously issue HELLO REQUEST objects. Each request is answered by an acknowledgment. Hello Messages also contain enough information so that one neighbor can suppress issuing hello requests and still perform neighbor failure detection. A Hello message may be included as a sub-message within a bundle message.

Neighbor state tracking is accomplished by collecting and storing a neighbor's state "instance" value. If a change in value is seen or if the neighbor is not properly reporting the locally advertised value, then the neighbor is presumed to have reset it's RSVP state. When communication is lost with a neighbor, then the instance value advertised to that neighbor is also changed. The HELLO objects provide a mechanism for polling for and providing an RSVP state instance value. A poll request also includes the sender's instance value. This allows the receiver of a poll to optionally treat the poll as an implicit poll response. This optional handling is an optimization that can reduce the total number of polls and responses processed by a pair of neighbors. In all cases, when both sides support the optimization the result will be only one set of polls and responses per failure detection interval. Depending on selected intervals, the same benefit can occur even when only one neighbor supports the optimization.

### 5.1. Hello Message Format

Hello Messages are always sent between two RSVP neighbors. The IP source address is the IP address of the sending node. The IP destination address is the IP address of the neighbor node.

HELLO messages SHOULD be exchanged between immediate RSVP neighbors. When HELLO messages are being the exchanged between immediate neighbors, the IP TTL field of all outgoing HELLO messages SHOULD be set to 1.

The Hello message format is as follows:

```
<Hello Message> ::= <Common Header> [ <INTEGRITY> ]
                          <HELLO>
```

For Hello messages, the Msg Type field of the Common Header MUST be

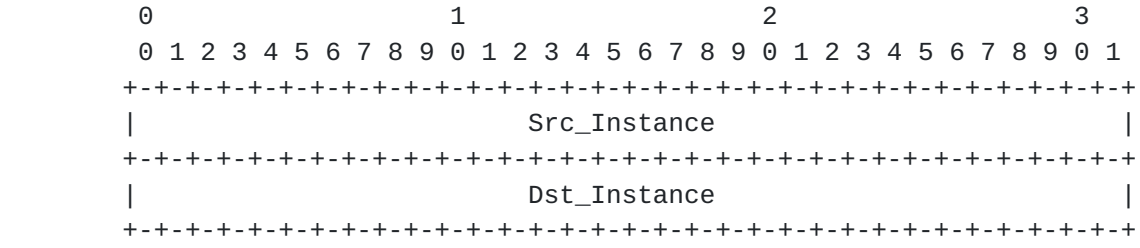
set to 14 (Value to be assigned by IANA).

## 5.2. HELLO Object

HELLO Class = 22 (Value to be assigned by IANA of form 0bbbbbbb)

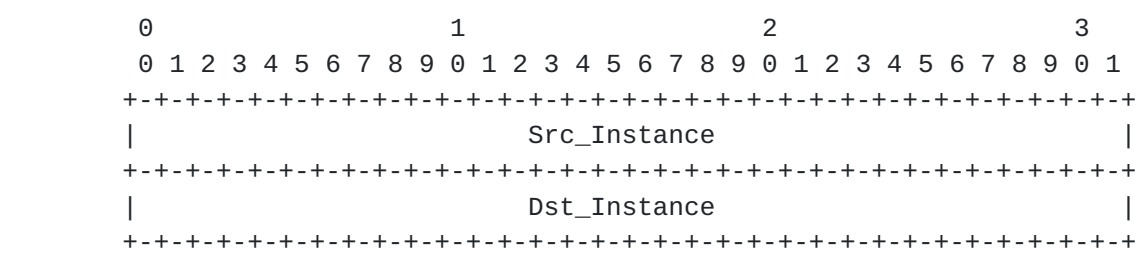
HELLO REQUEST object

Class = HELLO Class, C_Type = 1




HELLO ACK object

Class = HELLO Class, C_Type = 2




Src_Instance: 32 bits

a 32 bit value that represents the sender's RSVP agent's state. The advertiser maintains a per neighbor representation/value. This value MUST change when the agent is reset, when the node reboots, or when communication is lost to the neighboring node and otherwise remains the same. This field MUST NOT be set to zero (0).

Dst_Instance: 32 bits

The most recently received Src_Instance value received from the neighbor. This field MUST be be set to zero (0) when no value has ever been seen from the neighbor.

### 5.3. Hello Message Usage

A Hello message containing a HELLO REQUEST object MUST be generated for each neighbor who's state is being tracked. When generating a message containing a HELLO REQUEST object, the sender fills in the Src_Instance field with a value representing it's per neighbor RSVP agent state. This value MUST NOT change while the agent is maintaining any RSVP state with the corresponding neighbor. The sender also fills in the Dst_Instance field with the Src_Instance value most recently received from the neighbor. If no value has ever been received from the neighbor, a value of zero (0) is used. The generation of a message SHOULD be skipped when a HELLO REQUEST object was received from the destination node within the failure detection interval.

On receipt of a message containing a HELLO REQUEST object, the receiver MUST generate a Hello message containing a HELLO ACK object. The receiver SHOULD also verify that the neighbor has not reset. This is done by comparing the sender's Src_Instance field value with the previously received value. If the value differs, than the neighbor has reset and all state associated with the neighbor MUST be "expired" and cleaned up per standard RSVP processing. Additionally, all Path state advertised to the neighbor MUST be refreshed. The receiver SHOULD also verify that the neighbor is reflecting back the receiver's Instance value. This is done by comparing the received Dst_Instance field with the Src_Instance field value most recently transmitted to that neighbor. If the neighbor continues to advertise a wrong non-zero value after a configured number of intervals, then a node MUST treat the neighbor as if communication has been lost. In this case all state associated with the neighbor MUST be "expired" and cleaned up per standard RSVP processing. Additionally, the Src_Instance value advertised in the HELLO ACK object MUST be be different from the previously advertised value. This new value MUST continue to be advertised to the corresponding neighbor until a reset or reboot occurs, or until another communication failure is detected.

On receipt of a message containing a HELLO ACK object, the receiver MUST verify that the neighbor has not reset. This is done by comparing the sender's Src_Instance field value with the previously received value. If the value differs, than the neighbor has reset and all state associated with the neighbor MUST be "expired" and cleaned up per standard RSVP processing. Additionally, all Path state advertised to the neighbor MUST be refreshed. The receiver MUST also verify that the neighbor is reflecting back the receiver's Instance value. If the neighbor advertises a wrong value in the Dst_Instance field, then a node MUST treat the neighbor as if communication has been lost. In this case all state associated with the neighbor MUST be "expired" and cleaned up per standard RSVP

processing.

If no Instance values are received, via either REQUEST or ACK objects, from a neighbor within a configured number of failure detection intervals, then a node MUST presume that it cannot communicate with the neighbor. When this occurs, all state associated with the neighbor MUST be "expired" and cleaned up per standard RSVP processing, and all Path state advertised to the neighbor MUST be refreshed. If any state is removed or needs to be refreshed as a result of this case, then a new HELLO message MUST be immediately issued with a Src_Instance value different then the one advertised in the previous HELLO message. This new value MUST continue to be advertised to the corresponding neighbor until a reset or reboot occurs, or until another communication failure is detected.

## 5.4. Multi-Link Considerations

As previously noted, the Hello extension is targeted at detecting node failures not per link failures. When there is only one link between neighboring nodes or when all links between a pair of nodes fail, the distinction between node and link failures is not really meaningful and handling of such failures has already been covered. When there are multiple links shared between neighbors, there are special considerations. When the links between neighbors are numbered, then Hellos MUST be run on each link and the previously described mechanisms apply.

When the links are unnumbered, link failure detection MUST be provided by some means other than Hellos. Each node SHOULD use a single Hello exchange with the neighbor. When a node removes state due to a link failure, the node MUST advertise the removal of the state, via appropriate Tear messages, over a non-failed link. The case where all links have failed, is the same as the no received value case mentioned in the previous section.

## 5.5. Compatibility

The Hello extension is fully backwards compatible. The Hello class is assigned a class value of the form 0bbbbbbb. Depending on the implementation, implementations that don't support the extension will either silently discard Hello messages or will respond with an "Unknown Object Class" error. In either case the sender will fail to see an acknowledgment for the issued Hello. When a Hello sender does not receive an acknowledgment, it MUST NOT send MESSAGE_ID objects with the Last_Refresh flag set. This restriction will preclude neighbors from getting out of RSVP state synchronization.

Implementations supporting the Hello extension MUST also support the MESSAGE_ID extension and refresh suppression.

## 6. Reference Exponential Back-Off Procedures

This section is based on [Pan] and provides an example of how to implement exponential back-off. Implementations MAY choose to use the described procedures.

### 6.1. Outline of Operation

We propose the following feedback mechanism for exponential back-off retransmission of an RSVP message: When sending such a message, a node inserts a MESSAGE_ID object with the the ACK_Desired flag set. Upon reception, a receiving node acknowledges the arrival of the message by sending back an message acknowledgment (that is, a corresponding MESSAGE_ID ACK object.) When the sending node receives this message acknowledgment for a Path or Resv message, it will automatically scale back the retransmission rate for these messages for the flow. If the trigger message was for a different message type, no other further action is required.

Until the message acknowledgment is received, the sending node will retransmit the message. The interval between retransmissions is governed by a rapid retransmission timer. The rapid retransmission timer starts at a small interval which increases exponentially until it reaches a threshold. From that point on, the sending node will use a fixed timer to refresh Path and Resv messages and stop retransmitting other messages. This mechanism is designed so that the message load is only slightly larger than in the current specification even when the receiving node does not support message acknowledgment.

### 6.2. Time Parameters

The described procedures make use of the following time parameters. All parameters are per interface.

Rapid retransmission interval Rf:

Rf is the initial retransmission interval for unacknowledged messages. After sending the message for the first time, the sending node will schedule a retransmission after Rf seconds. The value of Rf could be as small as the round trip time (RTT) between a sending and a receiving node, if known. Unless a node

knows that all receiving nodes support echo-replies, a slightly larger configurable value is suggested.

Slow refresh interval Rs:

The sending node retransmits Path and Resv messages with this interval after it has determined that the receiving node will generate MESSAGE_ID ACK objects. To reduce the number of unnecessary retransmissions in a stable network, Rs can be set to a large value. The value of Rs should be configurable per each egress interface.

Fixed retransmission interval R:

A node retransmits the trigger message with the interval Rf*(1 + Delta)**i until the retransmission interval reaches the fixed retransmission interval R or a message acknowledgment has been received. If no acknowledgment has been received, the node continues to retransmit Resv and Path messages every R seconds. By default R should be the same value as the retransmission interval in the current RSVP specification.

Increment value Delta:

Delta governs the speed with which the sender increases the retransmission interval. The ratio of two successive retransmission intervals is (1 + Delta).

### 6.3. Example Retransmission Algorithm

After a sending node transmits a message containing a MESSAGE_ID object with the ACK_Desired flag set, it should immediately schedule a retransmission after Rf seconds. If a corresponding MESSAGE_ID ACK object is received earlier than Rf seconds, then retransmission SHOULD be canceled. Otherwise, it will retransmit the message after (1 + Delta)*Rf seconds. The staged retransmission will continue until either an appropriate MESSAGE_ID ACK object is received, or the retransmission interval has been increased to R. Once the retrans-mission interval has been increased to R, Path and Resv messages will be refreshed within the interval R. Other messages will not be retransmitted.

The implementation of exponential back-off retransmission is simple. A sending node can use the following algorithm after transmitting a message containing a MESSAGE_ID object with the ACK_Desired flag set:

```
if (Rk < R)  {
    Rk = Rk * (1 + Delta);
    retransmit the message;
    wake up after Rk seconds;
    exit;
}
else  {    /* no reply from receivers for too long: */
    Rk = R;
    if (message is a Path or Resv Message) {
        send out a refresh message;
        wake up after Rk seconds;
        exit;
    }
    else  {
        clean up any associated state and resources;
        exit;
    }
}
```

Asynchronously, when a sending node receives a corresponding MESSAGE_ID ACK object, it will change the retransmission interval Rk to Rs and, for non Path or Resv messages, clean up any associated state and resources.

## 7. Acknowledgments

This document represents ideas and comments from the MPLS-TE design team and participants in the RSVP Working Group's interim meeting. Thanks to Yoram Bernet, Fred Baker, Roch Guerin, Henning Schulzrinne, Andreas Terzis, David Mankins and Masanobu Yuhara for specific feedback on the document.

Portions of this work are based on work done by Masanobu Yuhara and Mayumi Tomikawa [Yuhara].

## 8. Security Considerations

No new security issues are raised in this document. See [RFC2205] for a general discussion on RSVP security issues.

## 9. References

[Awduche] Awduche, D. et al. Requirements for Traffic Engineering over MPLS, Internet Draft, draft-awduche-mpls-traffic-eng-00.txt, April 1998.

[Pan] Pan, P., Schulzrinne, H., "Staged Refresh Timers for RSVP," Global Internet'97, Phoenix, AZ, Nov. 1997. http://www.ctr.columbia.edu/~pan/papers/timergi.ps

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," RFC 2119.

[RFC2205] Braden, R. Ed. et al, "Resource ReserVation Protocol -- Version 1 Functional Specification", RFC 2205, September 1997.

[Yuhara] Yuhara, M., Tomikawa, M. "RSVP Extensions for ID-based Refreshes," Internet Draft, draft-yuhara-rsvp-refresh-00.txt, April 1999.

## 10. Authors' Addresses

Lou Berger
LabN Consulting
Voice: +1 301 468 9228
Email: lberger@labn.net

Der-Hwa Gan
Juniper Networks, Inc.
385 Ravendale Drive
Mountain View, CA 94043
Voice: +1 650 526 8074
Email: dhg@juniper.net

George Swallow
Cisco Systems, Inc.
250 Apollo Drive
Chelmsford, MA 01824
Voice: +1 978 244 8143
Email: swallow@cisco.com

Ping Pan
Bell Labs, Lucent
101 Crawfords Corner Road, Room 4C-508
Holmdel, NJ 07733
USA
Phone: +1 732 332 6744
Email: pingpan@dnrc.bell-labs.com